Achieving O(1) IP Lookup on GPU-based Software Routers

Jin Zhao, Xinya Zhang, Xin Wang, and Xiangyang Xue
School of Computer Science, Fudan University
Shanghai, China
{jzhao, 06300720198, xinw, xyxue}@fudan.edu.cn

ABSTRACT

IP address lookup is a challenging problem due to the increasing routing table size, and higher line rate. This paper investigates a new way to build an efficient IP lookup scheme using graphics processor units(GPU). Our contribution here is to design a basic architecture for high-performance IP lookup engine with GPU, and to develop efficient algorithms for routing prefix operations such as lookup, deletion, insertion, and modification. In particular, the IP lookup scheme can achieve $O(1)$ time complexity. Our experimental results on real-world route traces show promising 6x gains in IP lookup throughput.

Categories and Subject Descriptors

C.2.6 [**Computer Communication Networks**]: Internetworking - Routers

General Terms

Design

Keywords

Software router, GPU, IP lookup

1. INTRODUCTION

The core function of router's IP lookup engines is longest prefix matching (LPM),which determines the next-hop by comparing the incoming IP addresses against a set of stored prefixes in routing table. With CIDR (Classless Inter-Domain Routing), the prefix lengths may vary from 1 to 32 for IPv4 addresses. Due to the continuous growth in network link rates and routing table size, IP lookup engines are faced with enormous performance challenges. The lookup engines have to be able to answer ever-increasing number of lookup queries over a few hundred thousand routing prefixes. At the same time, IP lookup engines also have to accommodate demands for other update operations such as addition and deletion of prefixes, and the modification of next-hop for existing prefixes. TCAM-based hardware approach has the capability of parallelly searching all the prefixes simultaneously, leading to low access latency. However, TCAM also suffers from high cost due to high circuit density. Currently, there have been many efforts in scaling software routers using off-the-shelf, general-purpose PC [1]. Software routers usually employs trie structure for IP lookup, like the binary trie, and multibit trie. The main performance bottleneck for PC-based software routers is main memory access since LPM will invole additional memory accesses when traversing down the trie hierachy. There has been previous work in accelerating software router using Graphical Processing Units (GPUs) [3, 2]. By matching the incoming IP lookup requests in parallel on GPU, the overall throughput of IP lookup can be improved.

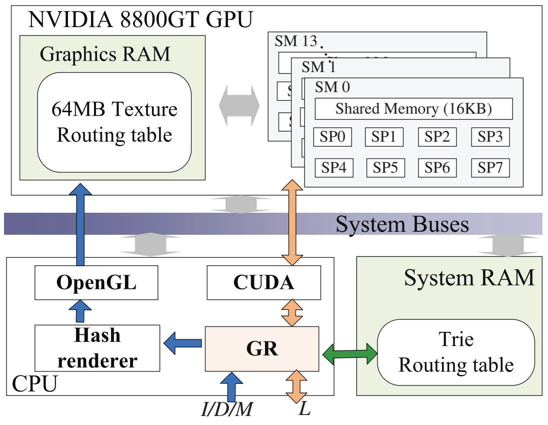

Figure 1: GPU-based IP lookup engine

In this paper, we investigate an alternative approach to building cost-effective and high-performance IP lookup engines and the corresponding routing table update schemes using GPUs. Besides the parallelism, we propose an IP lookup scheme that could achieve O(1) time complexity for each IP lookup. The key innovations include:(1)IP addresses are directly translated into memory addresses using a large routing table on GPU memory. (2)We also map the route-update operations by leveraging GPU's graphics processing facilities,like Z-buffer, Stencil buffer.

2. ARCHITECTURE

We now describe the proposed GPU-accelerated IP lookup architecture *GR* as shown in Fig. 1. We consider four kinds of routing operations: lookup (L), modification (M), insertion (I) and Deletion(D).

IP lookup. Observing that IP-lookup operations are far frequent than route-update operations, we seek to trade some performance in update for better lookup throughput. To this end, *GR* employs a large routing table on GPU, which enables simple translation from IP address to the memory address of matched entry. To speed up IP lookup, we can store all the possible prefix entries $2^{32} = 4G$ in GPU's memory. In this way, each IP lookup takes only $O(1)$ memory access. As the memory sizes of existing mainstream graphics cards are at an order of magnitude of 256MB or 512MB, in this paper, our implementation only considers up to 24 bits/16M prefix entries, which is not a very cost for modern GPUs.

In addition, with NVIDIA's CUDA, *GR* can simultaneously execute a group of IP lookup requests on GPU's many-core architecture.

However, directly mapped lookup adds to the complexity in route-update since there are correlations between the routing entries when using longest prefix matching. For example, if A.B.C/24 is updated, some prefixes A.B.D/x ($x < 24$) should be updated as well. Such update overheads are tremendous. In order to address the problem, we seek to use GPU's graphics render operations to implement route-updates.

From GPU's point of view, the large table which contains 2^{24} elements is a 4096×4096 texture which can be rendered and read. The first obstacle is that if we linearly map the IP address prefix "A.B.C" to memory address $A \cdot 65536 + B \cdot 256 + C$, the rect of a subnet can not form a square in the texture, which can be rendered in only one OpenGL command. To overcome this obstacle, a square hash mapping policy is proposed. This hash function accepts a 24-bit number as input, and output a 2D coordinate. Assuming the input is $b_0 b_1 b_2 \cdots b_{23}$, the output is

$$\begin{cases} x = b_0 b_2 b_4 \cdots b_{22} \\ y = b_1 b_3 b_5 \cdots b_{23} \end{cases}$$

This function is inspired from the quadtree in 2D space. Using this hash function, every prefix entry covers a square area in the texture, which can be efficiently manipulated by graphics processing facilities.

Modification. To modify the next-hop information of a given entry, a lookup operation will be performed first. After locating the memory address, *GR* will write the new next-hop information to the target memory.

Insertion. Another obstacle occurs while trying to insert a routing entry in a routing table which already contains some entries. Consider the following situation: A routing table has entries: A/8, and A.B.C/24, what would happen if inserting a new entry A.B/16? As a result, the A.B.C/24 would be incorrectly overwritten with the next-hop port of A.B/16. The problem can be overcomed by using Z-Buffer algorithm, while the rendering square shall have different depth value corresponding to it's prefix's length. At the same time, the current value stored in Z-Buffer represents this entry's prefix length. The process of adding a prefix entry is straightforward:(1)set the GL depth function to GL_GEQUAL, (2)Generate the square of the prefix, (3)Draw the square with depth=prefix's slash value.

Deletion. The last operation a router should support is to remove(or delete) a routing entry. Unfortunately, Z-Buffer facility supplied by OpenGL hardware is not powerful enough for removing a routing prefix. The stencil buffer is considered indeed. The operation needed by deleting an entry is as follows: replacing the value of deleted entry with its parent's. We draw it in two passes: in the first pass, the deleted square's stencil buffer is cleared to zero, in the next pass the deleted square's stencil buffer is set to its parent's depth value.

Table 1: Routing Prefix Operations

	Lookup(/ms)	Insert(/s)	Modify(/s)	Delete(/s)
GR(FUNET)	136,458	42,776.2	44,284.6	23,723.9
GR(RIS)	138,850	47,620.6	47,392.8	4,794.17
trie(FUNET)	19,372.9	3.30×10^6	5.98×10^6	2.09×10^6
trie(RIS)	36,850.7	6.05×10^6	15.3×10^6	5.79×10^6

3. PERFORMANCE

In this section, we present the experimental results for *GR* compared to GPU-accelerated trie-based schemes. We have two routing tables: FUNET [4] and RIS [5] . The traces is based on a real packet trace containing 99,840 entries from FUNET. The result is an average of 100 runs on a desktop with AMD Athlon 64 x2 4400+ 2.3GHz CPU, 1GB RAM and NVIDIA 8800GT GPU. The results in terms of lookup, insertion, deletion and modification speeds are given in Table 1. As can been seen from the table, the proposed scheme leads to significantly faster lookup (say, over 130,000 entries per *ms*) and the speedup could be up to 6 times better. We also observe that the trie-based scheme outperforms our *GR* in terms of update speeds, i.e. deletion, insertion, and modification. However, considering that the real-world routing update operations are not so frequent, say, about a few thousand BGP updates per second, the proposed schemes also suffice.

4. CONCLUSIONS

We present the design and evaluation of GPU-accelerated IP lookup engine that exploits the massive parallelism to speedup routing table lookup and show that,with proper desgin, there is the potential for significant improvement, e.g., 6x faster than trie-based implementation. We also designed efficient algorithms that can map deletion, insertion, and modification operations to graphics processing. We hope these preliminary results will encourage the development of GPU-based software routers.

5. ACKNOWLEDGMENTS

This work was supported in part by STCSM under grant 08dz150010E, by NSFC under grant 60803119, and by 863 program under grant 2009AA01A348.

6. REFERENCES

[1] M. Dobrescu, N. Egi, K. Argyraki, B. gon Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: Exploiting parallelism to scale software routers. In *ACM SOSP*, October 2009.

[2] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: Massively parallel packet processing with gpus to accelerate software routers. In *USENIX NSDI '10 poster*, April 2010.

[3] S. Mu, X. Zhang, N. Zhang, J. Lu, Y. S. Deng, and S. Zhang. Ip routing processing with graphic processors. In *DATE '10*, March 2010.

[4] FUNET. `http://www.nada.kth.se/~snilsson/`.

[5] RIS. `http://data.ris.ripe.net/`.